AD-A103 113 KANSAS STATE UNIV MANHATTAN DEPT OF COMPUTER SCIENCE F/G 9/2
RESEARCH IN FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPME--ETC(U)
DEC 76 P S FISHER, F J MARYANSKI DAAG29-76-G-0108
UNCLASSIFIED CS-76-20

1 OF 1
AD
A103113

END
DATE
FILMED
9-81
DTIC

CS-76

LEVEL III A103096 15

AIRMICS

Army Institute for Research in Management Information and Computer Science

313 Calculator Bldg.
GA Institute of Technology
Atlanta, GA 30332

Technical Report

AD A103113

RESEARCH IN FUNCTIONALLY DISTRIBUTED COMPUTER SYSTEMS DEVELOPMENT

Kansas State University

Virgil Wallentine
Principal Investigator

VOLUME VII

A USER--TRANSPARENT MECHANISM FOR THE DISTRIBUTION OF A CODASYL DATA BASE MANAGEMENT SYSTEM

U.S. ARMY COMPUTER SYSTEMS COMMAND FT BELVOIR, VA 22060

81 8 19 074

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) 9 Interim rept.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A103113	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 Research in Functionally Distributed Computer Systems Development, Volume VI. A USER-TRANSPARENT MECHANISM FOR THE DISTRIBUTION OF A CODASYL DATA BASE MANAGEMENT SYSTEM.		5. TYPE OF REPORT & PERIOD COVERED Iterim
		6. PERFORMING ORG. REPORT NUMBER 14 CS-76-22
7. AUTHOR(s) 10 Paul S. Fisher, Fred Maryanski, Virgil E. Wallentine		8. CONTRACT OR GRANT NUMBER(s) 15 DAAG 29-76-G-0108
9. PERFORMING ORGANIZATION NAME AND ADDRESS Kansas State University, Department of Computer Science, Manhattan, KS 66506		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 11
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Research Office, P O Box 12211, Research Triangle Park, NC 27700		12. REPORT DATE December 1976
		13. NUMBER OF PAGES 36 pages
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) US Army Computer Systems Command, Attn: CSCS-AT, Ft. Belvoir, VA 22060 12 39		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

DDBMS
DBMS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

-over-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

341132

-ABSTRACT-

A software organization is presented to provide for data definition and manipulation in a distributed data base management system. With the mechanism for distributing the data base proposed here, the physical location of the data is transparent to the user program. A Device Media Control Language is specified for the assignment of control of and access to a data base area to a set of processors. Procedures for reassignment of the control and access functions as well as the transfer of data between processors are provided. The basic hardware and software requirements for a computer network capable of supporting a distributed data base management system are discussed along with a specification of the software required for a processor in a distributed data base network.

A User-Transparent Mechanism for the Distribution of a CODASYL Data Base Management System[1]

TR CS 76-22

December, 1976

Fred J. Maryanski
Paul S. Fisher
Virgil E. Wallentine

[1]The work reported herein is supported by the U.S. Army Research Office Grant No. DAAD-29-76-G-0108.

Abstract

A software organization is presented to provide for data definition and manipulation in a distributed data base management system. With the mechanism for distributing the data base proposed here, the physical location of the data is transparent to the user program. A Device Media Control Language is specified for the assignment of control of and access to a data base area to a set of processors. Procedures for reassignment of the control and access functions as well as the transfer of data between processors are provided. The basic hardware and software requirements for a computer network capable of supporting a distributed data base management system are discussed along with a specification of the software required for a processor in a distributed data base network.

I. Introduction

This paper presents a software organization for a distributed data base management system (DDBMS). A DDBMS is a data base management system that resides on a network of computers. The processors in the network may perform any combination of the three following functions.

a. Front-end = act as user interface, receive input, transmit output.

b. Host = execute the application program.

c. Back-end = control data base access through execution of data base system software.

The DDBMS software structure presented in this paper reflects the CODASYL-type data base systems [1,2]. The basic software distribution and several possible hardware configurations for DDBMS systems are discussed in Reference [3]. The emphasis of this paper is to specify the software functions that are required in order to provide for proper data definition and manipulation in a DDBMS.

One of the principle tenets of the proposed DDBMS organization is that the physical distribution of the data be transparent to the user. This implies that at the application program level, both the program and the user are unconcerned with the precise physical location of the data or of the processor that is accessing the data. In addition, the DDBMS must have the capability of moving data among secondary storage devices and DBMS functions among processors. However, the user may stipulate that units of data be physically close. Additionally, it is necessary for the system software to be portable in order for an application program to execute on any host machine and access data through any back-end processor. This ability, to request relocating of tasks and data, is partially within the

DDBMS, but as with all application software, allocation of resources is dependent on the network operating system (NOS) upon which the DDBMS is constructed (in this paper it is supported by an NOS subsystem called Network Resource Control (NRC)). Information on the characteristics of a network operating system capable of supporting a DDBMS can be obtained in Reference [4].

The remaining sections of this paper contain a set of proposed mechanisms for defining and utilizing data in a DDBMS which has the capabilities described in the preceding paragraphs.

II. Data Definition

In a CODASYL-type DBMS the description of the data is carried out in three steps. Initially, the schema Data Description Language (DDL) is used to describe the logical organization and format of the data base. That portion of the data base accessible to a particular program is defined by means of the sub-schema DDL. The schema and sub-schema are both logical descriptions of the data. The logical to physical mapping is accomplished through use of the Device Media Control Language (DMCL). It is important to note that the CODASYL committee considered the DMCL as an implementation-dependent feature of a DBMS and consequently, has not specified a DMCL. It is through the DMCL that the distribution of the data base is accomplished.

In the definition of a distributed data base (as in the case of a central system) the existence or lack of physical proximity of records is determined by their placement in areas. Areas serve as the atomic data unit in terms of the distribution of the data base. The DMCL is used to associate areas with both logical back-end processors and secondary storage devices. This information is compiled from the DMCL and stored in the Area Logical Location (ALL) Table. The system may transport areas between physical devices. Such actions would remain transparent to the DBMS application programs provided the ALL tables and the tables of contents of the actual disk cartridges indicating, respectively, the physical location of the area and the contents of the cartridge are properly updated.

By restricting the effects of transparency to the description of the areas in the DMCL, both the schema and sub-schema DDL's remain unchanged in a DDBMS. The syntax of a DMCL for a distributed CODASYL-type data base system is shown in Figure 1.

The PAGE NUMBER and SIZE sentences in the Area Section are used to

```
SCHEMA SECTION

    DMCL FOR SCHEMA SCHEMANAME.

SUBSCHEMA SECTION

    SUBSCHEMA IS SUBSCHEMANAME.

AREA SECTION

    AREA IS AREANAME.
       NUMBER OF PAGES IS INTEGER1.
       PAGE SIZE IS INTEGER2.
       BACKEND IS PROCESSORNAME.
       DEVICE IS DEVICENAME.
          (TYPE IS IDENTIFIER.)
       HOSTS ARE (PROCESSORNAMES).
```

FIGURE 1

DMCL SYNTAX

describe the size of the area. The BACKEND sentence provides a logical name for the processor which controls access to the area. The DEVICE sentence provides a logical name and physical type for the device upon which the area is to be stored. A physical type can include disk, tape, cassette, floppy disk, or any other mass storage medium. The physical device type name can be dependent upon the back-end processor or (preferrably) follow a standard network convention. The HOSTS sentence logically names processors which may contain application programs that access the data in the area being described.

The HOSTS sentence provides an additional measure of security to the DDBMS. The data base administrator can specify if the area may be accessed globally or restricted to a certain application system. It is important to note that the HOSTS are identified logically in the DMCL. The HOSTS sentences are specifications as to which application function may access the data in an area. This application function may reside on several distinct processors and may be moved among processors by the network operating system. However, unless the data base administrator assigns a logical function name that appears in the HOSTS sentence of an area to a physical processor, no program executing on that processor may access data in that area.

The product of a DMCL compilation is an ALL Table whose entry format is depicted in Figure 2. An ALL Table is maintained for each schema. Within the table the information is grouped by area. The area information is obtained directly from the DMCL code.

It should be noted that processors and devices are identified logically in the ALL Table.. This allows areas and tasks to change their physical location in a manner transparent to the DBMS. The logical-to-physical mapping for processors is maintained in the network operating system nucleus on each machine. The companion mapping for devices is maintained locally at each

SCHEMA NAME	AREA NAME	BACKEND NAME	DEVICE NAME	PAGE SIZE	FIRST PAGE	LAST PAGE	PTR TO HOST* LIST

* HOST LIST IS A LINKED LIST OF HOSTS FOR THE AREA.

FIGURE 2

ALL TABLE ENTRY FORMAT

network node. Figure 3 illustrates the ALL Table and logical-to-physical map for a sample distributed data base.

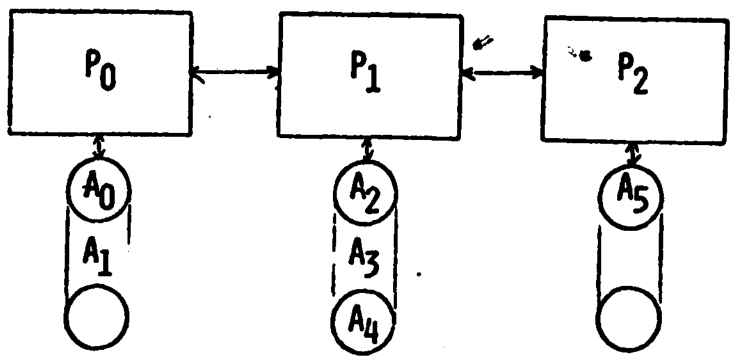
III. Data Manipulation in Distributed Data Access

The mechanisms for data manipulation in a DDBMS can be presented by considering the steps that are required for a user program to access data on a device connected to another processor.

Before an application program can access data in an area, it must first issue a READY statement [1] for that area. The data base system obtains the logical back-end processor and device name from the ALL Table entry of the area named in the READY statement. The physical processor name is obtained from Logical-to-Physical Processor Map maintained by the network operating system. A message is then sent to the back-end processor requesting that the area be made available to the application program and that a task be created for that sub-schema (if one does not already exist). The general format of the messages sent in the DDBMS is given in Figure 4. If the device containing the area is online at the back-end processor and all integrity and security requirements are satisfied, the back-end transmits a message to the host processor and the application program may begin to utilize the data in that area.

There are several situations which can hinder the completion of the READY statement. First, we treat the problem of the device(s) containing the area not being online. In this case, a message is transmitted to the operator of the back-end processor, requesting mounting of the proper disk pack. The pack is identified by the name obtained from the ALL Table. When the pack is placed online, the data base system automatically verifies its name against that requested by the application to detect any possible operator errors. Each data base contains a header which indicates the logical

DATA BASE (P_K - PROCESSOR, A_K - AREA)



ALL TABLE (PARTIAL)

AREA	BACKEND	HOSTS PTR	
A_0	B_0	$\rightarrow$	H_0 H_2
A_1	B_0	ALL	
A_2	B_1	$\rightarrow$	H_1
A_3	B_2	$\rightarrow$	H_0 H_2
A_4	B_2	$\rightarrow$	H_2
A_5	B_3	ALL	

L-P-P MAP

LOGICAL NAME	PHYSICAL ID
B_0	P_0
B_1	P_1
B_2	P_1
B_3	P_2
H_0	P_2
H_1	P_1
H_2	P_0

FIGURE 3 SAMPLE DISTRIBUTED DATA BASE

SEND MESSAGE (TO_ID, MESSAGE, SIZE, EVT_ID)
WHERE

TO_ID IS THE SYMBOLIC NAME OF THE TASK WHICH IS TO RECEIVE THE MESSAGE.

MESSAGE SPECIFIES THE BEGINNING ADDRESS OF THE MESSAGE TO BE SENT.

SIZE SPECIFIES THE SIZE OF THE MESSAGE TO BE SENT.

EVT_ID IS AN EVENT UNIQUELY IDENTIFIED WITH THIS MESSAGE.

RECEIVE MESSAGE (FROM_ID, MESSAGE, EVT_ID, SIZE)
WHERE

FROM_ID IDENTIFIES THE NAME OF A TASK FROM WHICH A MESSAGE IS TO BE RECEIVED.

EVT_ID, SIZE, AND MESSAGE ARE AS BEFORE.

WAIT (EVT_ID)
WHERE

EVT_ID IS AN EVENT WHICH IS SET TO "HAPPENED" WHEN ITS ASSOCIATED OPERATION IS COMPLETE. IT ALLOWS PROCESSES TO SUSPEND AWAITING PARTICULAR OPERATION.

SEND_COMMAND (TO_ID, COMMAND, EVT_ID)

ACCEPT_COMMAND (FROM_ID, COMMAND, EVT_ID)
WHERE COMMAND IS A FIXED SIZE (SMALL) MESSAGE

CONNECT (TO_ID, COMMAND)

DISCONNECT (FROM_ID, TYPE)
WHERE

$$\text{TYPE} = \begin{cases} \text{IMMEDIATE,} & \text{ABOUT ALL ACTIVE MESSAGES} \\ \text{QUIESCE,} & \text{ALLOW ACTIVE MESSAGES TO BE SENT BUT NO MORE INITIATED} \end{cases}$$

FIGURE 4

CONCEPTUAL MESSAGE FORMATS

(back-end) processors capable of accessing its data. This feature is included for security purposes. If the processor names correspond, the back-end processor will notify the host processor that the area is available for data manipulation.

Another problem occurs when there is no entry in the Logical-to-Physical Processor Map for the back-end processor name obtained from the ALL Table. In this case the host processor must obtain from the Data Dictionary a list of processors that have the potential of operating under that logical back-end name. Messages requesting that a back-end processor assume the functions associated with that logical name are then transmitted to the processors in that list. Once a back-end is identified, a request is made to mount the appropriate device for the area being READYed. If no machine can assume the requested logical processor role, the application program on the host processor is terminated.

When a processor assumes a back-end function, a task must be created in that processor to handle the data base requests for that area. All other processors in the network then are notified of the new logical name for the processor.

Once the area has been READYed, the user program on the host machine may issue DML commands to reference the data. The area name for any record to be accessed by the DML commands is available in the sub-schema which is attached to the application process during process initiation. Using the area name, the physical location of the record in the network can be determined from the ALL Table, the host machine then transmits a message to the appropriate back-end processor which performs the DBMS operation and transmits the results back to the host processor via the message system.

In a CODASYL-type DBMS, it is possible that an operation on a record in one area may result in the need for operations on records in other areas (for example, the removal of an owner record and thus all its members from the data base). In such situations, the back-end processor controlling the operation determines the back-end processor name for the effected area from the ALL Table. If the processor names are different, a message indicating the necessary data base action must be sent to the back-end processor of the area. This procedure could reoccur several times before completion of the original DBMS operation, depending upon the complexity and distribution of the data base. When the original back-end processor has received completion messages from all of the secondary back-end machines, it then transmits a message with the appropriate data and status information to the host computer.

IV. Task and Data Movement

A. Conceptual Aspects

In a DDBMS, it may be desirable for reasons of efficiency or security to change the physical location of a data base management task or data area. Movement of data can occur either logically by a programmed (file) transfer of an area between storage devices or physically by an operator moving a storage device from one computer to another. Tasks are moved in a DDBMS by making use of features provided in a network operating system.

The case of logical data movement will be considered first. When an area is moved between devices controlled by the same back-end processor, the device name must be modified in the ALL Table of that back-end processor and the header record on the storage media must be updated. Movement between devices attached to different processors requires updating the ALL Tables for all back-end processors that control any portion of the sub-schemas containing the transported areas. It can be seen from the description of data manipulation in Section III that if the ALL Tables are properly updated, there will be no effect on either the user program on the host or the data base system on the back-end machine.

The logical movement of an area is accomplished via network operating system utility programs, U_s and U_r, executing on B_s, the sending back-end processor, and B_r, the receiving back-end processor, respectively. The following procedure is executed to move area A from B_s to B_r.

Procedure 1 (Logical Area Movement)

Let SST_a be the set of the sub-schema tasks for all sub-schemas containing area A.

1. U_s notifies all tasks in SST_a on B_s that A will be moved.

2. All tasks in SST_a will notify the message utility not tc accept any further message for DML's in area A.

3. The message system on B_s instructs the message utility on all hosts accessing this area that any DML messages referring to A are held in the host machine by the message utility.

4. Each task in SST_a will notify U_s when all DML's for A are complete.

5. U_s sends a message to U_r indicating that A is ready to be moved.

6. U_s writes A onto secondary storage thus placing the latest version of A onto disk.

7. A is transferred from secondary storage of B_s to secondary storage of B_r.

8. If for any sub-schema A was the only area controlled by B_s, then U_s must remove the entries for that sub-schema from the ALL Table maintained by B_s.

9. If for any sub-schema task SST_a A was the only area controlled by B_s, then the task for that sub-schema must be destroyed on B_s.

10. If A is the only area for a given sub-schema on B_r, then U_r must update the ALL Tables of B_r to include the entries for that sub-schema.

11. Any sub-schema tasks on B_s destroyed by the movement of A must be

created (if not already present) on B_r.

12. If by receiving A, B_r is now assuming a new logical back-end function, then the L-P-P Map on B_s must be updated.

13. If by sending A to B_s, B_r no longer retains that particular back-end function, then the L-P-P Map on B_s must be updated.

14. Update all L-P-P Maps in the network if necessary.

15. The sub-schema tasks in SST_a on B_r obtain the list of hosts for A from the ALL Table.

16. The sub-schema tasks in SST_a notify the message utilities which are queueing requests from their corresponding host tasks that they will now accept DML messages for A.

Figure 5 gives a precedence graph for Algorithm 1. The interaction of the utility tasks and system software resulting from the execution of Procedure 1 is illustrated in Figure 6. Only the software directly involved in logical area movement is pictured in Figure 6.

The situation in which a physical device is moved is more complex than logical movement. The most difficult situation occurs if the pack (or any other type of storage device) is active on a back-end processor and the operator indicates to the system the desire to remove that pack from online status. The data base management software on the back-end processor must complete all requests to areas on that pack and inform the effected hosts to hold all new requests for those areas. The operator may then disengage the pack and remount it on a new machine. When the pack is mounted on the new machine, the procedure outlined in Section III for bringing up a new pack in the DDBMS can be followed. It is important to note that in order for a processor to be eligible to receive the transferred pack, it must be read and write compatible with the source processor as well as have the necessary logical processor function assigned to it.

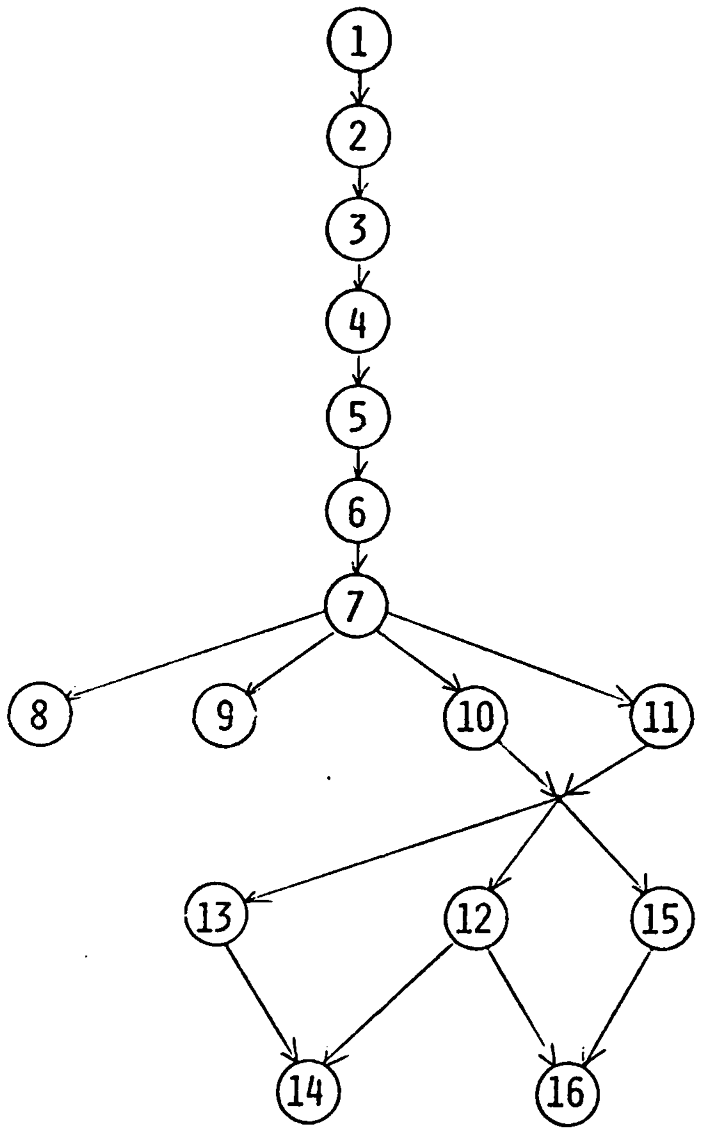


FIGURE 5

PRECEDENCE GRAPH FOR PROCEDURE 1

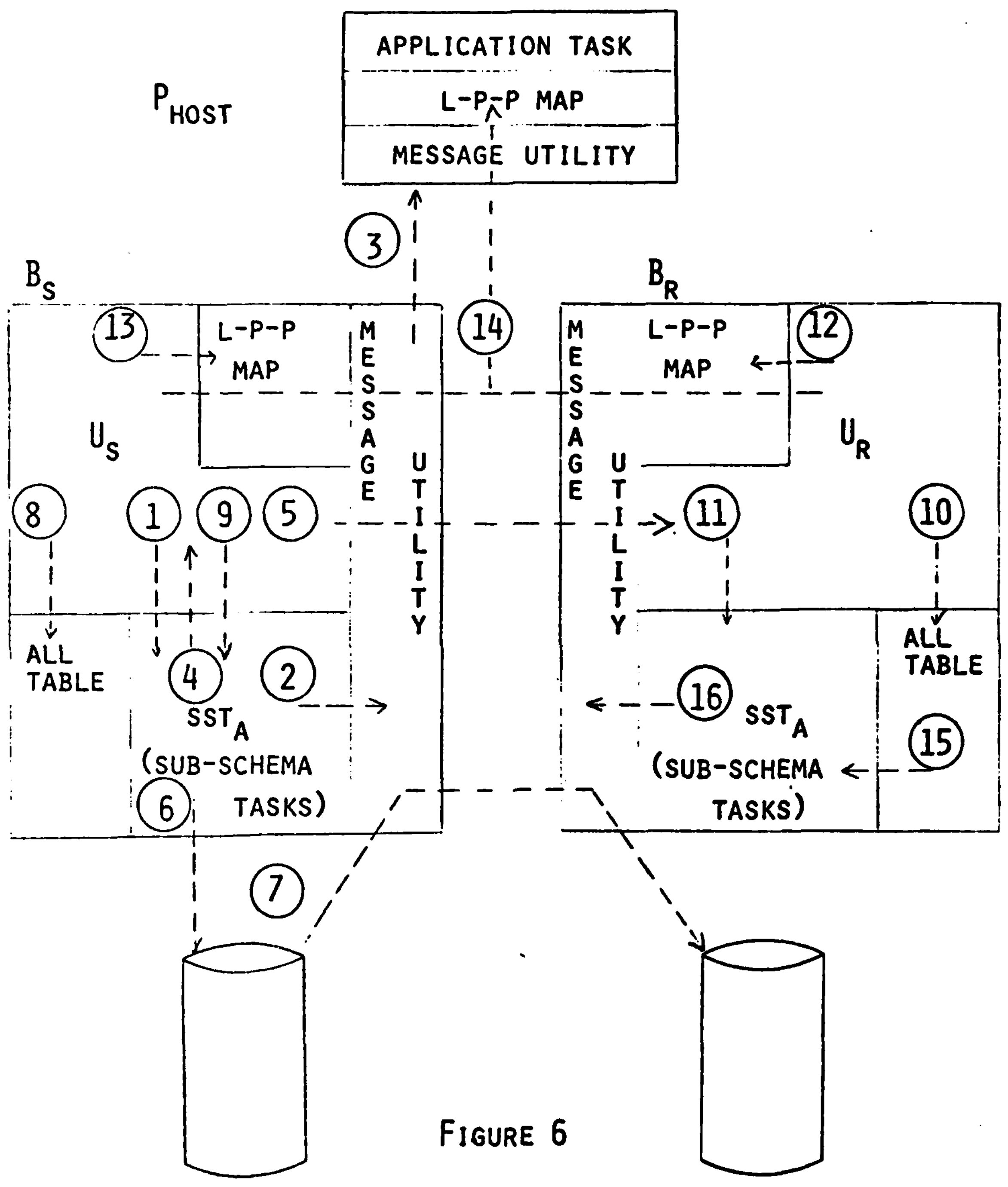


FIGURE 6

LOGICAL AREA MOVEMENT

In order to allow the DDBMS to stop processing a pack prior to its movement, the list of active areas on the pack is determined from the Device Header List that is maintained by the back-end processor. For each area, the list of logical host names is obtained from the ALL Table. The back-end proeessor then accesses its Logical-to-Physical Processor Map to determine the host computers which must be sent the messages indicating an area transfer.

The back-end data base software which controls the mounting of a new pack must be cognizant of requirements stating that in order for a given pack to be accessed some other pack (or packs) must also be attached to the same processor. Such constraints could occur in the cases of multi-device areas or for security or efficiency reasons.

The physical movement of a data base pack between back-end processors results from carrying out the steps of Procedure 2. A pack shutdown utility U_d is first executed on the sending back-end processor B_s. A pack mount utility, U_m, is then executed on B_r, the receiving back-end processor.

Procedure 2 (Physical Pack Movement)

1. The B_s operator activates pack shutdown utility, U_d.
2. U_d initiates steps 1 - 4,6,8-9, and 13 of Procedure 1 for each area on the pack.
3. U_d notifies the B_s operator that the pack may be moved.
4. The B_s operator removes the pack.
5. B_r operator mounts the pack.
6. B_r operator activates pack mount utility, U_m.
7. U_m initiates steps 10-12 and 14-16 of Procedure 1 for each area on the pack.

There are two types of task movement in a DDBMS; the transferring of an application program between processors, and the interprocessor movement of back-end software. Both cases can be considered as processor function reassignment.

In the case of host function movement the only action required is a change in the Logical-to-Physical Processor Map in the network operating system. The redefinition of the ability for appropriate processors to access an area can also be accomplished by executing the DMCL compiler with a modified HOSTS paragraph.

The back-end functions may only be transferred if the receiving processor is linked to the storage device(s) containing the area. The mechanisms for accomplishing the transfer are identical to the host situation. Either the network operating system Logical-to-Physical Processor Map or the DMCL BACK-END sentence can be amended.

Thus as in the case of data movement, tasks (processor functions) can be moved in a DDBMS with minimal overhead and with no alteration required to user or DBMS software. This statement is predicated upon the portability of the data base system software. Given the state of the industry, movement must be restricted to homogeneous machines for the present.

One important fact concerning the relationship between logical and physical names for physical entities (processors, devices) is that the mapping can be one-to-one, many-to-many, one-to-many, or many-to-one. A one-to-many mapping indicates a multi-processor configuration or an area spread across several devices. A processor or storage device can be identified by several logical names, thus producing the many-to-one relationship. The many-to-many mapping is a merger of the two previously mentioned situations. The flexibility of the logical to physical mapping provides the data base administrator with considerable latitude in the distribution of the data base.

Figure 7 depicts a one-to-many device mapping. The area NAMES is spread over three physical devices. The record occurrences, FRED, VIRG, and PAUL, all reside on separate disk packs controlled by the same back-end processor. The multi-device area concept is found in many commercially available data base management systems.

A one-to-many processor mapping implies the existence of a multi-processor back-end. A multi-processor back-end consists of several processors joined via a memory-to-memory connection. Each processor has access to the areas controlled by the back-end function shared by the processors. If a processor in a multi-processor back-end configuration does not have a direct physical connection, it requests that a processor having such a link perform the I/O transfers. With a shared memory interprocessor connection, such requests are performed at machine memory speeds. Figure 8 illustrates a multi-processor back-end configuration. The concept of multi-processor back-end machines is discussed by Lowenthal in Reference [5]. Figure 9 portrays a many-to-many mapping which is realized by combining the configurations shown in the two previous figures.

For a given distributed data base system, the range of a multi-processor back-end configuration is limited by the network topology. A generalized network such as MIMICS [4] is composed of a collection of machine clusters joined together. Within each cluster several processor nodes linked via high speed memory connections, (5 - 10 Megabyte/sec). Figure 10 shows the general topology of the MIMICS network.

The following rules apply to multi-processor back-ends:

1. The processors comprising a multi-processor back-end must be members of the same cluster.

2. Areas may not span clusters. As shown in Figures 8 thru 10, it is

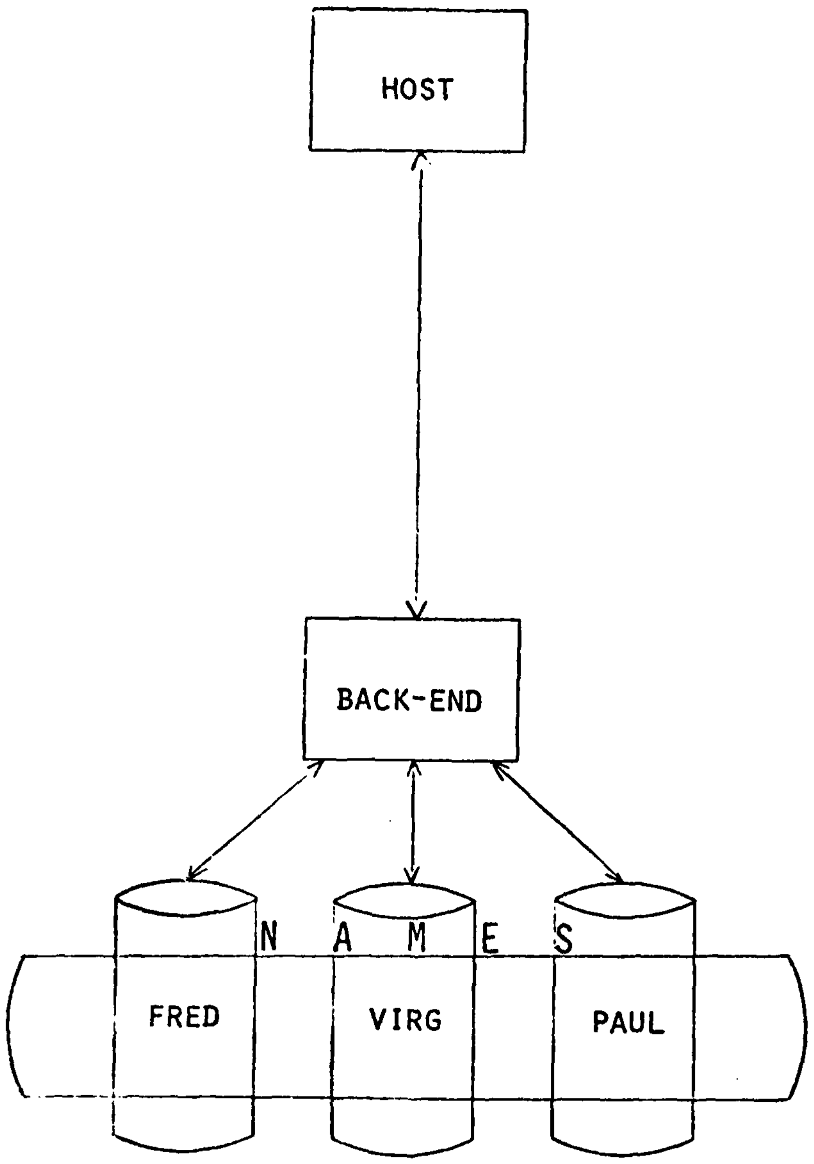


FIGURE 7
MULTI-DEVICE AREA

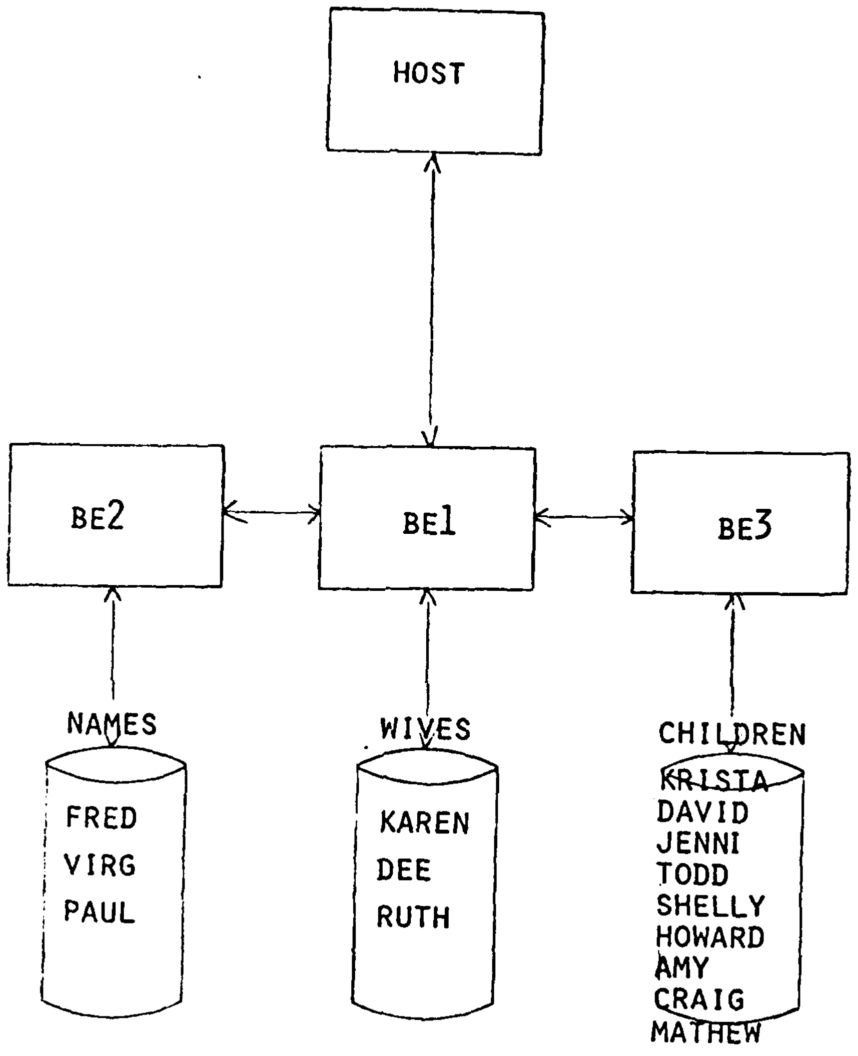


FIGURE 8

MULTI-PROCESSOR BACK-END

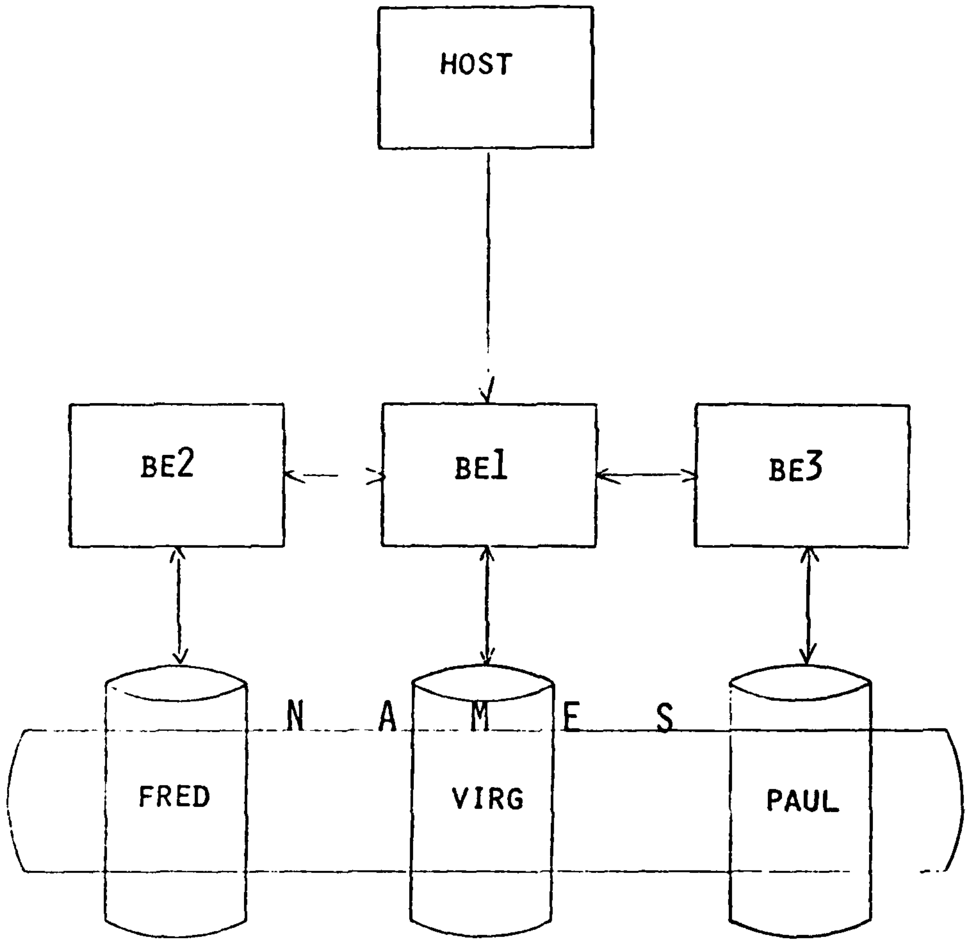


FIGURE 9

MULTI-DEVICE AREA WITH MULTI-PROCESSOR BACK-END

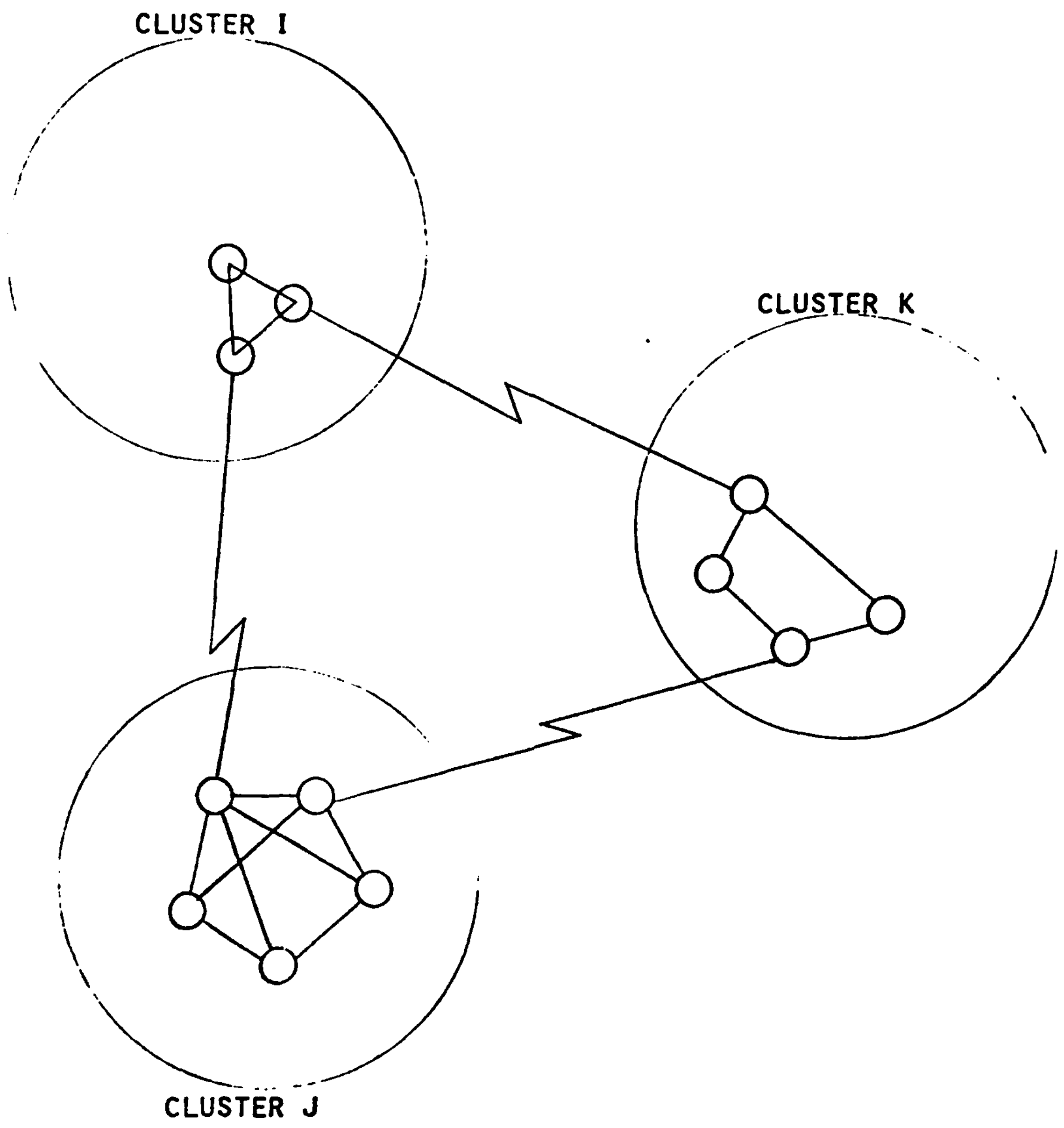


FIGURE 10

GENERAL TOPOLOGY OF MIMICS

not necessary for each node in a multi-processor back-end cluster to have a communication link to a host computer (or a cluster of host computers). The node with the communication link (BE1 in Figures 8 and 9, for example) functions as the master back-end for that multi-processor back-end configuration. The master handles the communication operations and parcels out the DML commands to be executed by the individual back-end processors.

An analysis of the performance, security integrity, and economic benefits of the multi-processor back-end concept has been performed by Lowenthal [5].

B. The Inter-Process Communication System (IPCS) of MIMICS

Movement of (areas of) data in a DDBMS is accomplished via some inter-process communication (message) utility which has the following functions:

1) It makes the topology of the network transparent to the application program;

2) it makes data distribution transparent to the application program;

3) it synchronizes the tasks which exchange data to insure no data is lost, garbled, or pilfered;

4) it manages the names of network tasks;

5) and finally, it transmits data and commands between tasks (application program and DBMS tasks, for example). The concepts of the IPCS commands available to a task level are provided in Figure 4.

The IPCS of MIMICS is connection-based. That is, the general scenario of IPCS usage by a task is as follows:

1. CONNECT (, , ---)
2. Exchange data using SEND and RECEIVE; exchange commands using SEND_COMMAND and ACCEPT_COMMAND; and WAIT on a particular function completion when necessary

3. DISCONNECT (, , ---)

The CONNECT/DISCONNECT functions establish and destroy data/command paths between tasks. The SEND/RECEIVE functions exchange data. The IPCS synchronizes the message SEND's from the source task and the RECEIVE's from the destination task to assure proper space is allocated for the message. The SEND_/ACCEPT_COMMAND functions allow commands to be exchanged between user tasks without prior data space allocation necessary. This permits user tasks to exchange simple commands to establish a protocol for exchanging messages. It also permits priority traffic for error or control information to be exchanged. This is the mechanism used for movement of areas in a transparent manner. Since all of the above SEND's and RECEIVE's do not force the task to stop execution until completion (to provide overlap of execution and data exchange), the task may choose to WAIT until a later time on a particular SEND or RECEIVE (event).

The general structure of the IPCS is shown in Figure 11. The application program conceptually exchanges DML's and data/status with the DBMS task. The User Envelope transmits these elements across a network as data and commands. The User Envelope thus maps source and destination tasks onto source and destination logical processors and then onto physical processors via the L-P-P map and the ALL table. Thus the System Envelope has the capacity to re-route DML's and data/status via a new mapping. The Message System merely does the movement of data between tasks which have established a connection.

Figure 12 illustrates the method by which the movement of an area between two back-end machines can be achieved in a manner transparent to an application program. It is assumed the application program task (APT) and the DBMS have an established connection. The general procedure, from the view of the message system, to move an area is as follows:

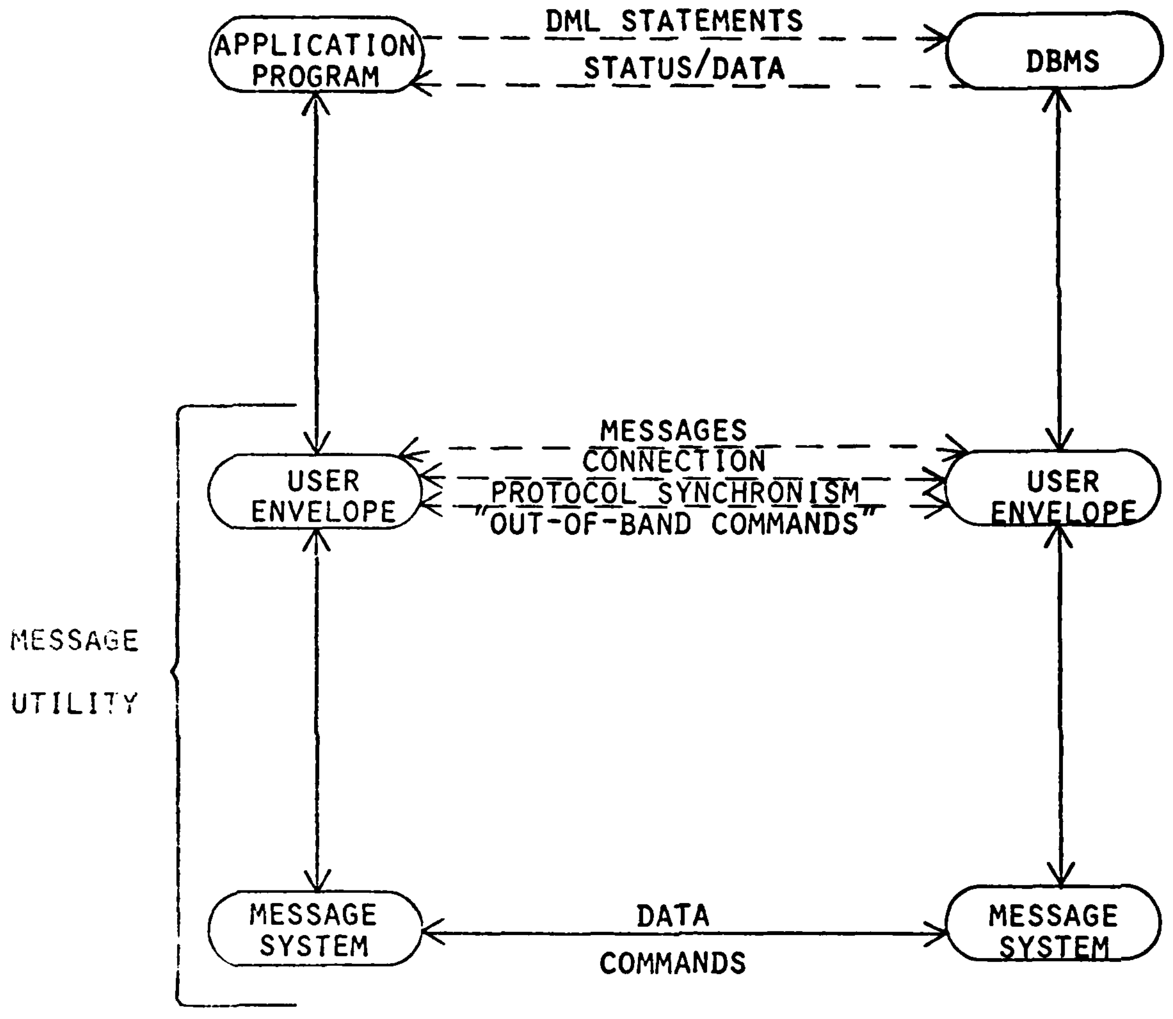


FIGURE 11

LEVELS OF MIMICS PROTOCOL

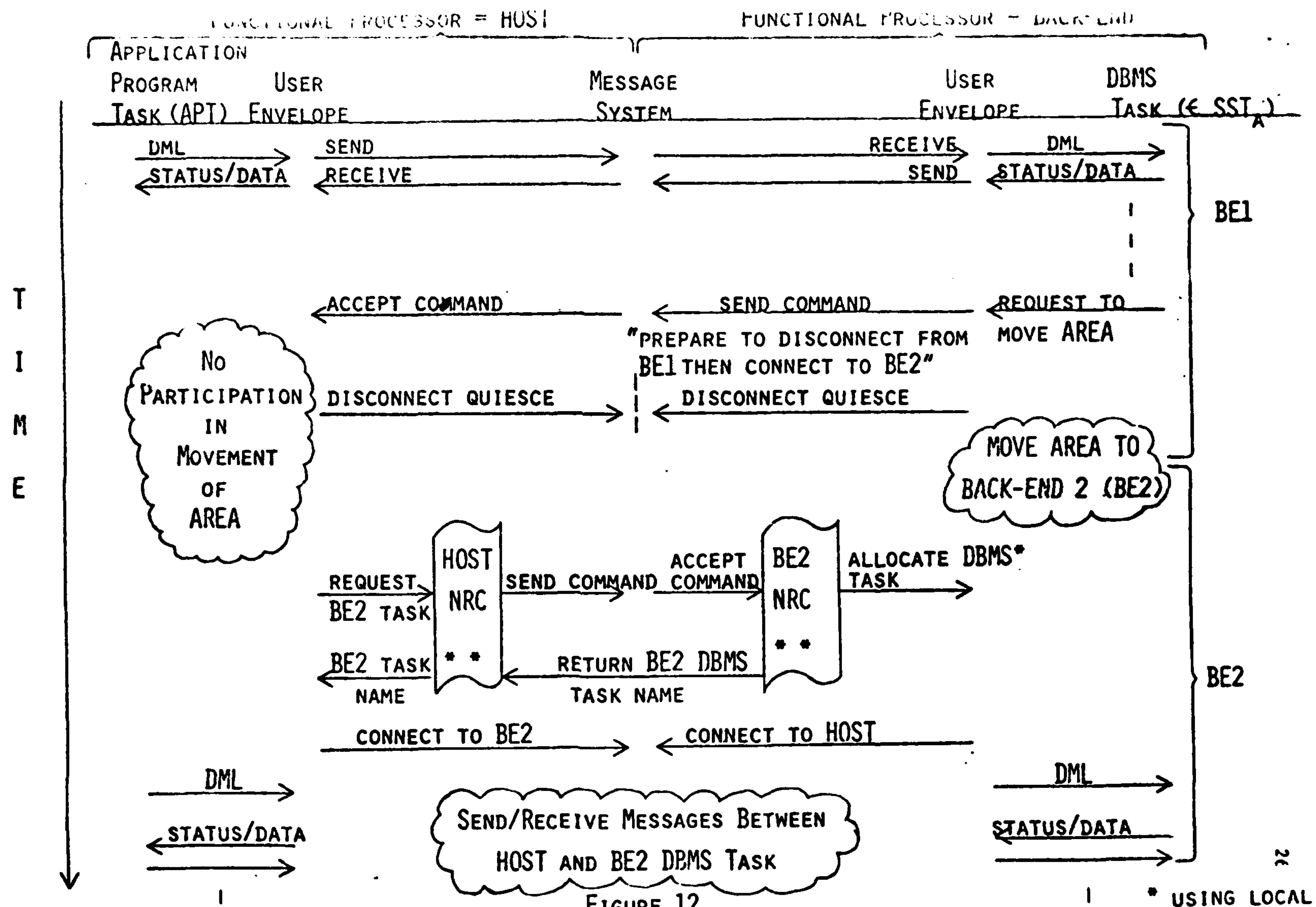


FIGURE 12

SCENARIO OF AREA MOVEMENT USING MIMICS PROTOCOL

* USING LOCAL OPERATING SYSTE

** NRC--NETWOF RESOURCE CONTRC

1) The User Envelopes exchange the desire (initiated from either end) to DISCONNECT after a certain sequence of messages is complete.

2) Each does a DISCONNECT (,QUIESCE). (These are synchronized by the Message System.)

3) The AREA is transferred by a file transfer protocol [6] via a connection between tasks U_s and U_r. (See Figure 6.)

4) The User Envelopes on the Host and Back-End 2 (BE2) then connect as follows:

The Host User Envelope requests a network task name for a DBMS task on BE2 from the Network Resource Controller (NRC) on the Host. The NRC's of all processors are fully connected. Thus this request is made of the NRC on BE2. It responds with the task name. The User Envelope then connects to the BE2 User Envelope and DML's and data/status flow between the Host APT and the BE2 DBMS task.

It is important to note that NRC must access only the L-P-P map. It should be clear from Figure 12 that the Host APT does not participate in the movement or knowledge of the movement of the AREA other than by observing some performance change.

V. Structure of Host and Back-End Software

A host computer in a DDBMS must contain software to execute application programs, to select the proper back-end for data base functions, and to communicate with the back-end processor. In order to meet these ends, a host must have a software organization similar to that depicted in Figure 13. The Logical-to-Physical Processor Map (L-P-P Map) is simply an array of physical identifiers indexed by logical names. As mentioned in prior sections, this map is used in the selection of the back-end processor. The Inter-Process Communication System (IPCS) [4] serves as a generalized communication system for the data base network.

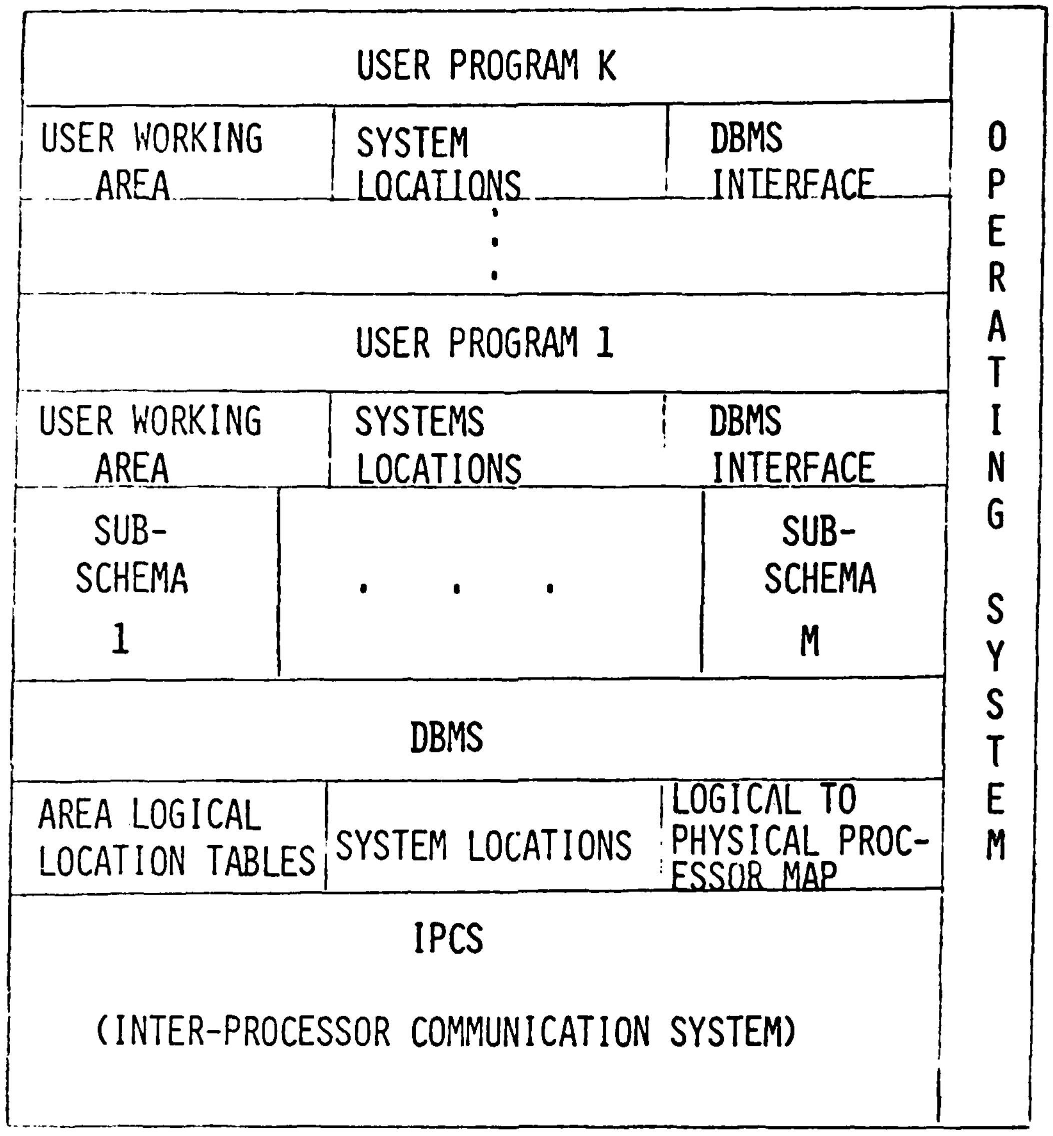


FIGURE 13

HOST SOFTWARE ORGANIZATION

A back-end processor must hold the sub-schema and DMCL Tables for each area in its domain. Due to the possibility of multiple back-end participation in DML execution, each back-end processor must contain complete ALL Tables for all sub-schemas containing areas managed by that machine. The back-end must also contain the Logical-to-Physical Processor Map in order to determine physical processor identification. As shown in Figure 14, the back-end software must also include a Device Header List (D.H. List) which indicates the contents of the secondary storage units. The Device Header List contains a linked list of areas with each list headed by a device indicator name.

It is important to note that a single machine may be both a host and back-end processor. In that situation, the software shown in both Figures 13 and 14 must reside upon that machine. If a processor assumes more than one logical host or back-end function, the DMCL Tables and sub-schemas for each logical processor function must be stored in the memory of the machine. Figure 15 illustrates the software required on processor P_1 of Figure 3. P_1 has two back-end functions, B_1 and B_2, and one host function, H_1, associated with it.

VI. The Data Dictionary

Each processor in a DDBMS may require access to any schema, sub-schema, or ALL Table. Copies of this information are maintained in the data dictionary. When a DBMS application task is loaded onto a host processor, its associated sub-schema and ALL Tables are obtained from the data dictionary. The data dictionary also holds the potential Logical-to-Physical Processor Map which indicates those processors permitted to assume a particular logical name. This list is used whenever an application program requests an area for which there is no active processor performing that back-end function. The general structure of a DDBMS data dictionary is illustrated in Figure 16.

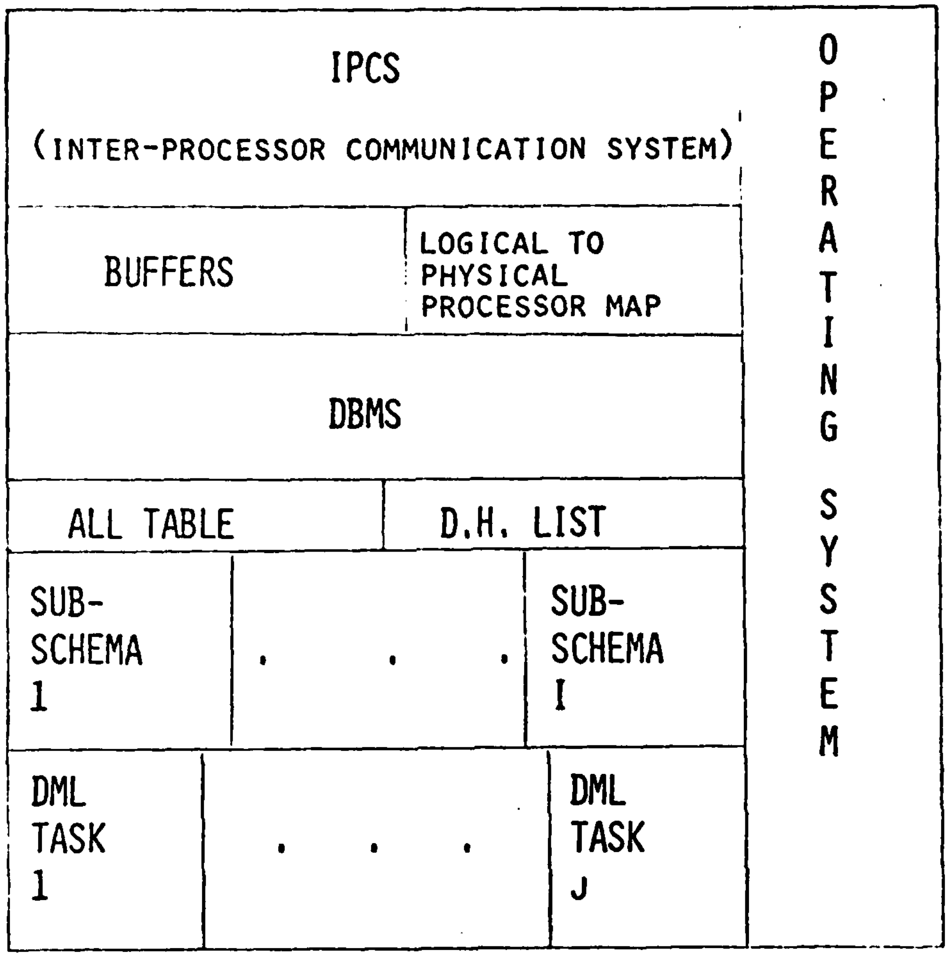


FIGURE 14

BACK-END SOFTWARE ORGANIZATION

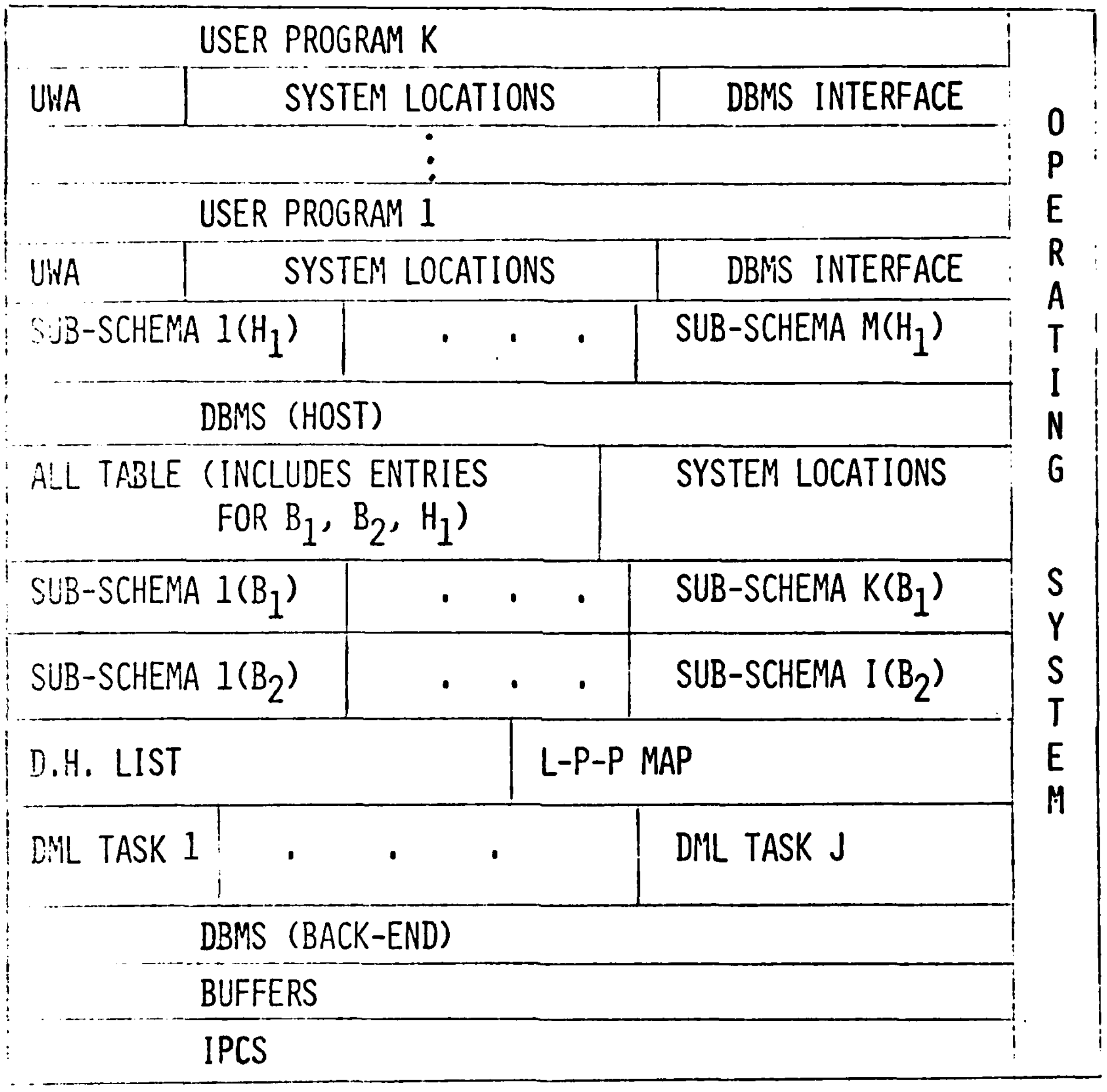


FIGURE 15

SOFTWARE ORGANIZATION FOR A PROCESSOR WITH MULTIPLE FUNCTIONS

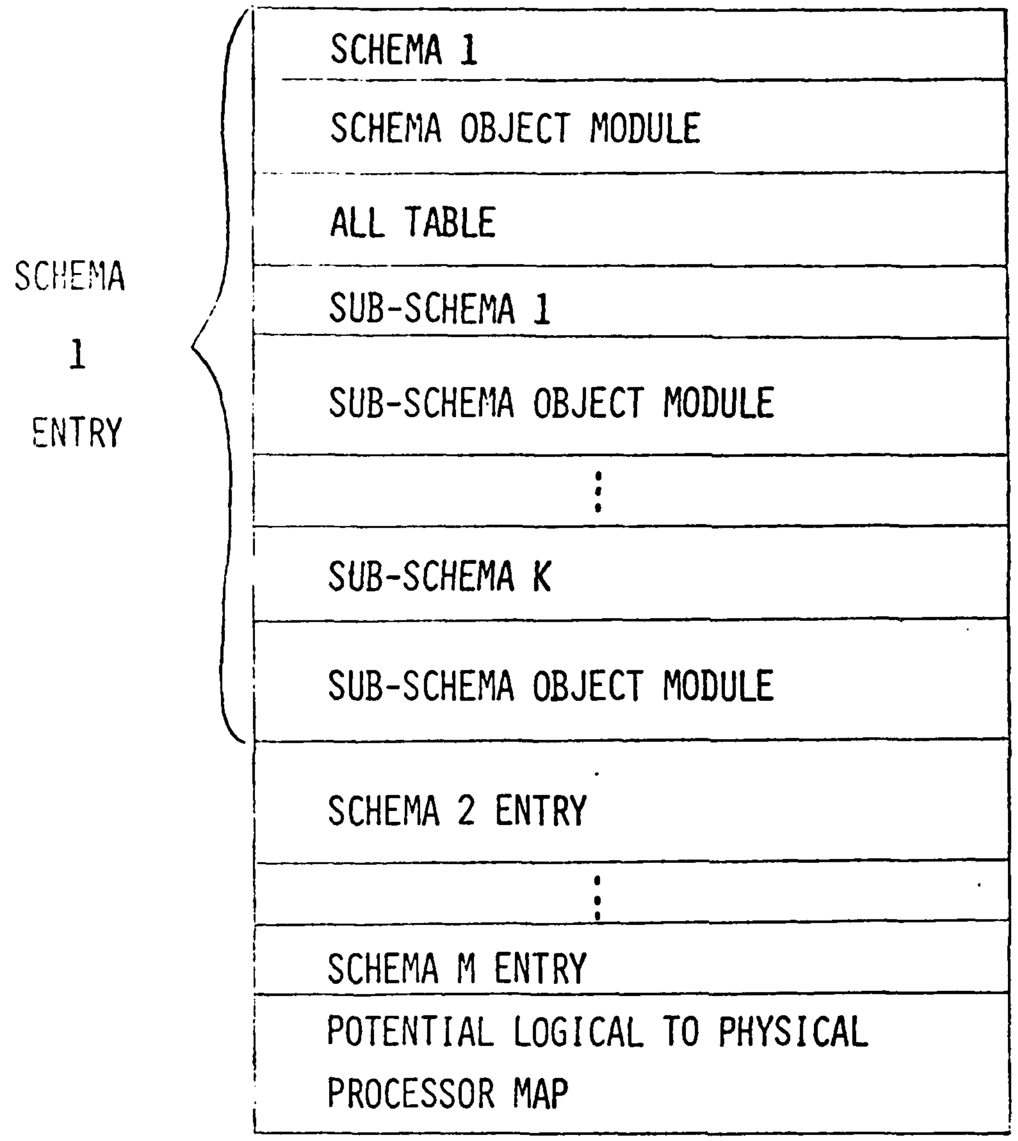


FIGURE 16

DATA DICTIONARY ORGANIZATION

The exact mechanism of implementing the data dictionary can vary with the structure of the underlying computer network. The dictionary must reside on a high speed secondary storage device for rapid access. The data dictionary is used generally for query with little update. Therefore, multiple copies of the data dictionary may be maintained for the sake of reliability and more rapid reference. In certain environments, it may be desirable to partition the data dictionary into sub-dictionaries which contain information on data residing in particular sections of the network.

VII. Conclusion

This paper has presented a mechanism for the distribution of a CODASYL-type data base management system in a manner that is transparent to the application program. The software structure presented herein presupposes an underlying computer network with the necessary hardware and software to allow interprocessor communication via a standardized message system. The basis for distribution in the DDBMS is the ALL Table which provides information on the location of each data base area.

The mechanisms detailed here provide a DDBMS communication facility that is relatively easy to realize. However, many of the problems of distributed data systems, as outlined by Fry and Sibley [7], still require practical solutions. The dilemmas posed by deadlock, backup, recovery, and security are extremely complex. Another formidable stumbling block for distributed data base systems is the general lack of portability and compatibility within both the hardware and software environments. The system described here could be implemented with moderate effort on homogeneous networks. For heterogeneous networks, advances in software portability and standardized communication protocols are required. Progress is being made in these areas although it is hampered somewhat by the marketing philosophy of "locking the user in" to a vendor's product line.

Bibliography

1. CODASYL COBOL Journal of Development, Dept. of Supply and Services, Material Data Management Branch, Ottawa, Ontario KIA OS5, (revised to) June, 1976.
2. CODASYL Data Description Language Journal of Development, Document C1362:113, U.S. Government Printing Office, Washington, D.C., 1973.
3. Maryanski, F.J., et al., "A Minicomputer Based Distributed Data Base System, Proc. IEEE-NBS Symposium on Trends and Application 1976: Micro and Mini Systems, May, 1976, pp. 113-118.
4. Wallentine, V.E., "MIMICS-The Capabilities to be Developed", Computer Science Dept., Kansas State University, Manhattan, KS. 66506, May, 1976.
5. Lowenthal, E.I., "The Backend Computer", MRI Systems Corp., P.O. Box 9968, Austin, Texas 78766, Apr., 1976.
6. Wallentine, V.E., et al., "Progress Report on Functionally Distributed Computer Systems Developemnt: Software and Systems Structure" TR CS77-4, Computer Science Dept., Kansas State University, Manhattan, KS. 66506, Dec. 1976.
7. Fry, J.P. and Sibley, E.H., "Evolution of Data-Base Management Systems", Computing Surveys, Vol. 8, No. 1, May, 1976, pp. 7-42.